



Instituto Politécnico Nacional

ESCUELA SUPERIOR DE COMPUTO

Ingeniería en Sistemas Computacionales



Compiladores

Práctica 3

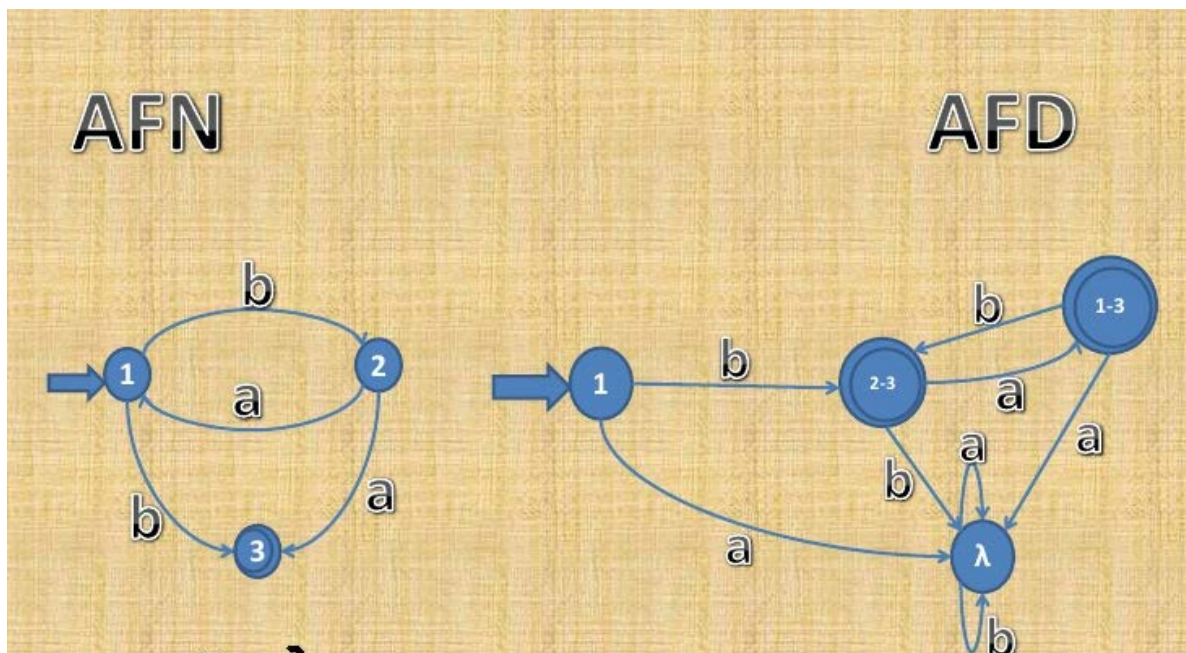
Algoritmo de los subconjuntos

PROFESOR: RAFAEL NORMAN SAUCEDO DELGADO

ALUMNO: DANIEL MURGUÍA JIMÉNEZ

No. boleta: 2016630491

GRUPO: 3CV6



Introducción

El algoritmo de los subconjuntos se emplea para efectuar la conversión de un autómata finito no determinista (AFN) a un autómata finito determinista (AFD) que acepte el mismo lenguaje que el AFN. Este algoritmo se apoya de 2 funciones fundamentales, las cuales son ϵ -cerradura(T) y mover(T,a), que se encargan de obtener todos los estados a los que se puede acceder con transiciones ϵ dado uno o un grupo de estados y que obtiene el conjunto a los que se puede acceder dado un conjunto de estados y un símbolo, respectivamente.

Desarrollo

Para el desarrollo de esta práctica se pidió implementar el algoritmo de los subconjuntos utilizando las clases AFN y AFD anteriormente creadas en la práctica 1. Debido a que no se completó de manera satisfactoria la práctica 1, se optó por recodificar las clases AFN y AFD en Python.

Para el desarrollo de este algoritmo fue necesario implementar los métodos ϵ -cerradura() y mover, de tal manera que fuera posible utilizarlos en conjunto con las clases AFN y AFD.

Para la implementación del método ϵ -cerradura(), se siguieron los pasos descritos en el libro “Compiladores Principios Técnicas y Herramientas”, que nos dice que se debe de tomar un conjunto de estados, agruparlos en un conjunto cerradura e introducir los estados en una pila. Una vez en la pila se procede a iterar a través de los estados de la pila buscando transiciones ϵ del estado en cuestión, si se encuentra una y el estado final de esa transición no se encuentra en el conjunto cerradura, se agrega a este.

```
1. def e_cerradura(self, estados):
2.     Tr = self.afn.transiciones
3.     cerradura = []
4.     cerradura.append(estados)
5.     pila = []
6.     pila.append(estados)
7.     while len(pila) != 0:
8.         estado = pila.pop()
9.         for transicion in Tr:
10.            if transicion.inicio == estado and transicion.simbolo == 'E':
11.                if transicion.fin not in cerradura:
12.                    cerradura.append(transicion.fin)
13.                    pila.append(transicion.fin)
14.     return cerradura
```

La implementación del método mover() fue más sencilla puesto a que este se encarga de obtener un conjunto de estados a los cuales se pueden llegar con un determinado símbolo, a partir de un conjunto de estados.

```
1. def mover(self, estados, simbolo):
2.     Tr = self.afn.transiciones
3.     m = []
4.     for estado in estados:
5.         for transicion in Tr:
```

```

6.         if estado == transicion.inicio and transicion.simbolo == simbolo:
7.             m.append(transicion.fin)
8.         return m

```

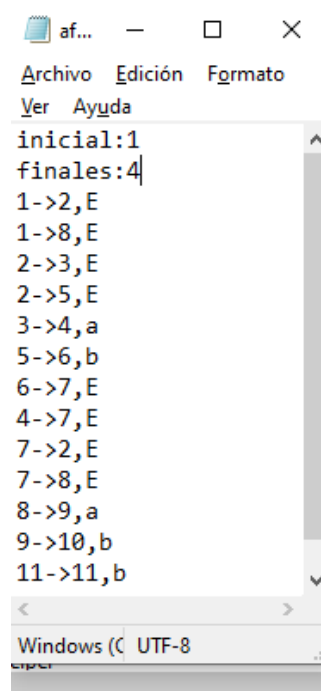
Finalmente se creo un método `construir_subconjuntos()` el cual es el encargado de la conversión de AFN a AFD, la clase `subconjuntos` al momento de instanciarse, recibe un AFN con el cual se pretende trabajar, al llamar a este método, se obtiene el conjunto `e_cerradura()` del estado inicial del AFN y se anexa a un diccionario “Destados” con el que se etiqueta con una llave cada subconjunto de estados que se encuentre. Los conjuntos de estados se encuentran marcado por un valor booleano, el cual nos ayuda a saber en que momento parar la iteración de subconjuntos, cuando todos los subconjuntos se encuentran marcados por un valor booleano “True”, se detiene. Para obtener los subconjuntos, se utilizan las dos funciones `e_cerradura` y `mover en conjunto`.

```

1. def construir_subconjuntos(self):
2.     etiqueta = 65
3.     se_encuentra = False
4.     self.Destados[str(chr(etiqueta))]=(self.e_cerradura(self.afn.inicial))
5.     self.Destados[str(chr(etiqueta))].sort()
6.     self.Destados[str(chr(etiqueta))].append(False)
7.     while not self.checar_marcados():
8.         self.Destados[str(chr(etiqueta))][-1] = True
9.         for simbolo in self.afn.alfabeto:
10.            U = self.e_cerradura(self.mover(self.Destados[str(chr(
11.                etiqueta))][-1],simbolo))
12.            U.sort()
13.            for key,val in self.Destados.items():
14.                if all(item in U for item in val):
15.                    se_encuentra = True
16.            if not se_encuentra:
17.                etiqueta += 1
18.                self.Destados[str(chr(etiqueta))]= U
19.                self.Destados[str(chr(etiqueta))].append(True)
20.                self.afd.agregar_transicion(chr(etiqueta-1),chr(etiqueta),
21.                    simbolo)

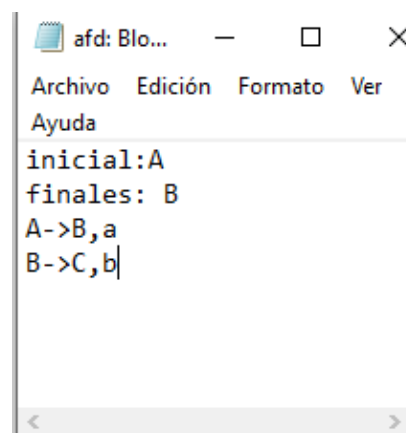
```

Para probar el funcionamiento de las clases, se introdujo el siguiente archivo con la definición de un autómata no determinista:



```
af...
Archivo Edición Formato
Ver Ayuda
inicial:1
finales:4
1->2,E
1->8,E
2->3,E
2->5,E
3->4,a
5->6,b
6->7,E
4->7,E
7->2,E
7->8,E
8->9,a
9->10,b
11->11,b
Windows (C UTF-8
```

Y de salida tras la ejecución del algoritmo se obtuvo la siguiente definición de autómata determinista:



```
afd: Blo...
Archivo Edición Formato Ver
Ayuda
inicial:A
finales: B
A->B,a
B->C,b
```

Conclusión

El desarrollo de esta práctica fue una buena oportunidad para entender más a fondo de que manera se puede obtener un AFD a partir de un AFN. Al codificarlo en Python en lugar de JAVA, se facilitó demasiado puesto a que el manejo de archivos en Python es de manera más directa y se pueden obtener los valores necesarios para operar de una manera más fácil y rápida.

Código de clase Subconjuntos

```
1. import AFD,AFN,Transicion
2.
3. class Subconjuntos(object):
4.     def __init__(self, afn):
5.         self.Destados = {}
6.         self.afn = afn
7.         self.afd = AFD.AFD()
8.
9.
10.    def e_cerradura(self, estados):
11.        Tr = self.afn.transiciones
12.        cerradura = []
13.        cerradura.append(estados)
14.        pila = []
15.        pila.append(estados)
16.        while len(pila) != 0:
17.            estado = pila.pop()
18.            for transicion in Tr:
19.                if transicion.inicio == estado and transicion.simbolo == 'E':
20.                    if transicion.fin not in cerradura:
21.                        cerradura.append(transicion.fin)
22.                        pila.append(transicion.fin)
23.        return cerradura
24.
25.
26.    def mover(self,estados,simbolo):
27.        Tr = self.afn.transiciones
28.        m = []
29.        for estado in estados:
30.            for transicion in Tr:
31.                if estado == transicion.inicio and transicion.simbolo == simbolo:
32.                    m.append(transicion.fin)
33.        return m
34.
35.
36.    def construir_subconjuntos(self):
37.        etiqueta = 65
38.        se_encuentra = False
39.        self.Destados[str(chr(etiqueta))]=(self.e_cerradura(self.afn.inicial))
40.        self.Destados[str(chr(etiqueta))].sort()
41.        self.Destados[str(chr(etiqueta))].append(False)
42.        while not self.checar_marcados():
43.            self.Destados[str(chr(etiqueta))][-1] = True
44.            for simbolo in self.afn.alfabeto:
45.                U = self.e_cerradura(self.mover(self.Destados[str(chr(
46.                    etiqueta))][-1],simbolo))
```

```

47.         U.sort()
48.         for key,val in self.Destados.items():
49.             if all(item in U for item in val):
50.                 se_encuentra = True
51.             if not se_encuentra:
52.                 etiqueta += 1
53.                 self.Destados[str(chr(etiqueta))]= U
54.                 self.Destados[str(chr(etiqueta))].append(True)
55.                 self.afd.agregar_transicion(chr(etiqueta-1),chr(etiqueta),
56.                                             simbolo)
57.         for key,val in self.Destados.items():
58.             if self.afn.inicial in val[:-1]:
59.                 self.afd.establecer_inicial(key)
60.             for item in self.afn.finales:
61.                 if item in val[:-1]:
62.                     self.afd.establecer_final(key)
63.
64.
65.     def checar_marcados(self):
66.         marcados = True
67.         for key, val in self.Destados.items():
68.             if val[-1] == False:
69.                 marcados = False
70.         return marcados
71.
72.     def obtener_afd(self):
73.         return self.afd
74.
75. if __name__ == "__main__":
76.     autn = AFN.AFN()
77.     autn.cargar_desde("afn.afn")
78.     autn.cargar_transiciones()
79.     autn.obtener_inicial()
80.     autn.obtener_finales()
81.     autn.obtener_alfabeto()
82.     subconjuntos = Subconjuntos(autn)
83.     subconjuntos.construir_subconjuntos()
84.     autd = AFD.AFD()
85.     autd = subconjuntos.obtener_afd()
86.     autd.guardar_en("afd")

```

Código de clase AFD en Python

```

1. import re
2. import Transicion
3.
4. class AFD(object):
5.     def __init__(self):
6.         self.inicial= ""
7.         self.finales = []
8.         self.transiciones = []
9.         self.definicion = ""
10.
11.
12.     def cargar_desde(self,fname):
13.         ##abrimos el archivo

```

```

14.         f = open(fname,encoding='utf-8')
15.         ##obtenemos el texto
16.         self.definicion = f.read()
17.         f.close()
18.
19.     def list_string(self, s):
20.         str1 = ""
21.         for ele in s:
22.             str1 += " " + str(ele)
23.         return str1
24.
25.     def guardar_en(self,fname):
26.         self.definicion = "inicial:{}\nfinales:{}".format(self.inicial,
27.                                                             self.list_string(
28.                                                                 self.finales))
29.         for tr in self.transiciones:
30.             atr = tr.obtener_atributos()
31.             self.definicion += "\n{}->{},{}".format(atr[0], atr[1],atr[2])
32.         f = open(fname+".afd",'w',encoding = 'utf-8')
33.         f.write(self.definicion)
34.         f.close()
35.
36.     def obtener_lenguaje(self):
37.         simbolos=[]
38.         for linea in self.definicion.splitlines():
39.             simbolo=""
40.             if re.match(r"(\d+)->(\d+),([a-zA-Z])",linea):
41.                 res = re.match(r"(\d+)->(\d+),([a-zA-Z])",linea)
42.                 simbolo = res.group(3)
43.                 simbolos.append(simbolo)
44.             simbolos = list(dict.fromkeys(simbolos))
45.             print(simbolos)
46.
47.     def cargar_transiciones(self):
48.         for linea in self.definicion.splitlines():
49.             inicio=0
50.             fin=0
51.             simbolo=""
52.             if re.match(r"([A-Z]?)->([A-Z]?),([a-z])",linea):
53.                 res = re.match(r"([A-Z]?)->([A-Z]?),([a-z])",linea)
54.                 inicio = res.group(1)
55.                 fin = res.group(2)
56.                 simbolo = res.group(3)
57.                 tr = Transicion.Transicion(inicio,fin,simbolo)
58.                 self.transiciones.append(tr)
59.
60.     def agregar_transicion(self,inicio,fin,simbolo):
61.         tr = Transicion.Transicion(inicio,fin,simbolo)
62.         self.transiciones.append(tr)
63.
64.     def eliminar_transicion(self,inicio,fin,simbolo):
65.         tr = Transicion.Transicion(inicio,fin,simbolo);
66.         indice = 0
67.         for transicion in self.transiciones:
68.             if transicion.igual(tr):
69.                 self.transiciones.pop(indice)
70.                 return
71.                 indice += 1
72.
73.     def obtener_inicial(self):

```

```

74.         lineas = self.definicion.splitlines()
75.         ini_str = re.match(r"([a-zA-Z]+):(\d+)", lineas[0])
76.         self.inicial = ini_str.group(2)
77.
78.     def obtener_finales(self):
79.         lineas = self.definicion.splitlines()
80.         ini_str = re.match(r"([a-zA-Z]+):(\s*\d+)", lineas[1])
81.         finales = ini_str.group(2).split(" ")
82.         for final in finales:
83.             self.finales.append(final)
84.
85.     def establecer_inicial(self, inicial):
86.         self.inicial = inicial
87.
88.     def establecer_final(self, final):
89.         self.finales.append(final)
90.
91.     def es_AFN(self):
92.         for transicion in self.transiciones:
93.             if transicion.obtener_atributos()[2] == 'E':
94.                 return True
95.         return False
96.
97.     def es_AFD(self):
98.         for transicion in self.transiciones:
99.             if transicion.obtener_atributos()[2] == 'E':
100.                 return False
101.         return True
102.
103.     """
104.     def acepta(self, cadena):
105.     def generar_cadena(self)
106.     """

```

Código de clase AFN en Python

```

1. import re
2. import Transicion
3.
4. class AFN(object):
5.     def __init__(self):
6.         self.inicial=0
7.         self.finales = []
8.         self.transiciones = []
9.         self.definicion = ""
10.        self.alfabeto=[]
11.
12.
13.    def cargar_desde(self, fname):
14.        ##abrimos el archivo
15.        f = open(fname, encoding='utf-8')
16.        ##obtenemos el texto
17.        self.definicion = f.read()
18.        f.close()
19.
20.    def obtener_alfabeto(self):

```



```

21.     simbolos=[]
22.     for linea in self.definicion.splitlines():
23.         simbolo=""
24.         if re.match(r"(\d+)->(\d+),([a-zA-Z])",linea):
25.             res = re.match(r"(\d+)->(\d+),([a-zA-Z])",linea)
26.             simbolo = res.group(3)
27.             simbolos.append(simbolo)
28.     simbolos = list(dict.fromkeys(simbolos))
29.     simbolos.remove('E')
30.     self.alfabeto = simbolos
31.
32.
33.     def list_string(self, s):
34.         str1 = ""
35.         for ele in s:
36.             str1 += " " + str(ele)
37.         return str1
38.
39.     def guardar_en(self,fname):
40.         self.definicion = "inicial:{}\n".format(self.inicial,
41.                                                self.list_string(
42.                                                    self.finales))
43.         for tr in self.transiciones:
44.             atr = tr.obtener_atributos()
45.             self.definicion += "\n{}->{},{}".format(atr[0], atr[1],atr[2])
46.         f = open(fname+".afn",'w',encoding = 'utf-8')
47.         f.write(self.definicion)
48.         f.close()
49.
50.     def cargar_transiciones(self):
51.         for linea in self.definicion.splitlines():
52.             inicio=0
53.             fin=0
54.             simbolo=""
55.             if re.match(r"(\d+)->(\d+),([a-zA-Z])",linea):
56.                 res = re.match(r"(\d+)->(\d+),([a-zA-Z])",linea)
57.                 inicio = int(res.group(1))
58.                 fin = int(res.group(2))
59.                 simbolo = res.group(3)
60.                 tr = Transicion.Transicion(inicio,fin,simbolo)
61.                 self.transiciones.append(tr)
62.
63.     def agregar_transicion(self,inicio,fin,simbolo):
64.         tr = Transicion.Transicion(inicio,fin,simbolo)
65.         self.transiciones.append(tr)
66.
67.     def eliminar_transicion(self,inicio,fin,simbolo):
68.         tr = Transicion.Transicion(inicio,fin,simbolo);
69.         indice = 0
70.         for transicion in self.transiciones:
71.             if transicion.igual(tr):
72.                 self.transiciones.pop(indice)
73.                 return
74.             indice += 1
75.
76.     def obtener_inicial(self):
77.         lineas = self.definicion.splitlines()
78.         ini_str = re.match(r"([a-zA-Z]+):(\d+)",lineas[0])
79.         self.inicial = int(ini_str.group(2))
80.

```

```

81.     def obtener_finales(self):
82.         lineas = self.definicion.splitlines()
83.         ini_str = re.match(r"([a-zA-Z]+):(\s*\d+)+", lineas[1])
84.         finales = ini_str.group(2).split(" ")
85.         for final in finales:
86.             self.finales.append(int(final))
87.
88.     def establecer_inicial(self, inicial):
89.         self.inicial = inicial
90.
91.     def establecer_final(self, final):
92.         self.finales.append(final)
93.
94.     def es_AFN(self):
95.         for transicion in self.transiciones:
96.             if transicion.obtener_atributos()[2] == 'E':
97.                 return True
98.         return False
99.
100.    def es_AFD(self):
101.        for transicion in self.transiciones:
102.            if transicion.obtener_atributos()[2] == 'E':
103.                return False
104.        return True
105.
106.    '''
107.    def acepta(self, cadena):
108.    def generar_cadena(self)
109.    '''

```

Código de clase Transición en Python

```

1.  class Transicion(object):
2.      def __init__(self, inicio, fin, simbolo):
3.          self.inicio = inicio
4.          self.fin = fin
5.          self.simbolo = simbolo
6.
7.      def obtener_atributos(self):
8.          return self.inicio, self.fin, self.simbolo
9.
10.     def igual(self, tr):
11.         return True if (self.inicio == tr.inicio and self.fin == tr.fin
12.                        and self.simbolo == tr.simbolo) else False
13.
14.     def __repr__(self):
15.         return self.inicio, self.fin, self.simbolo

```